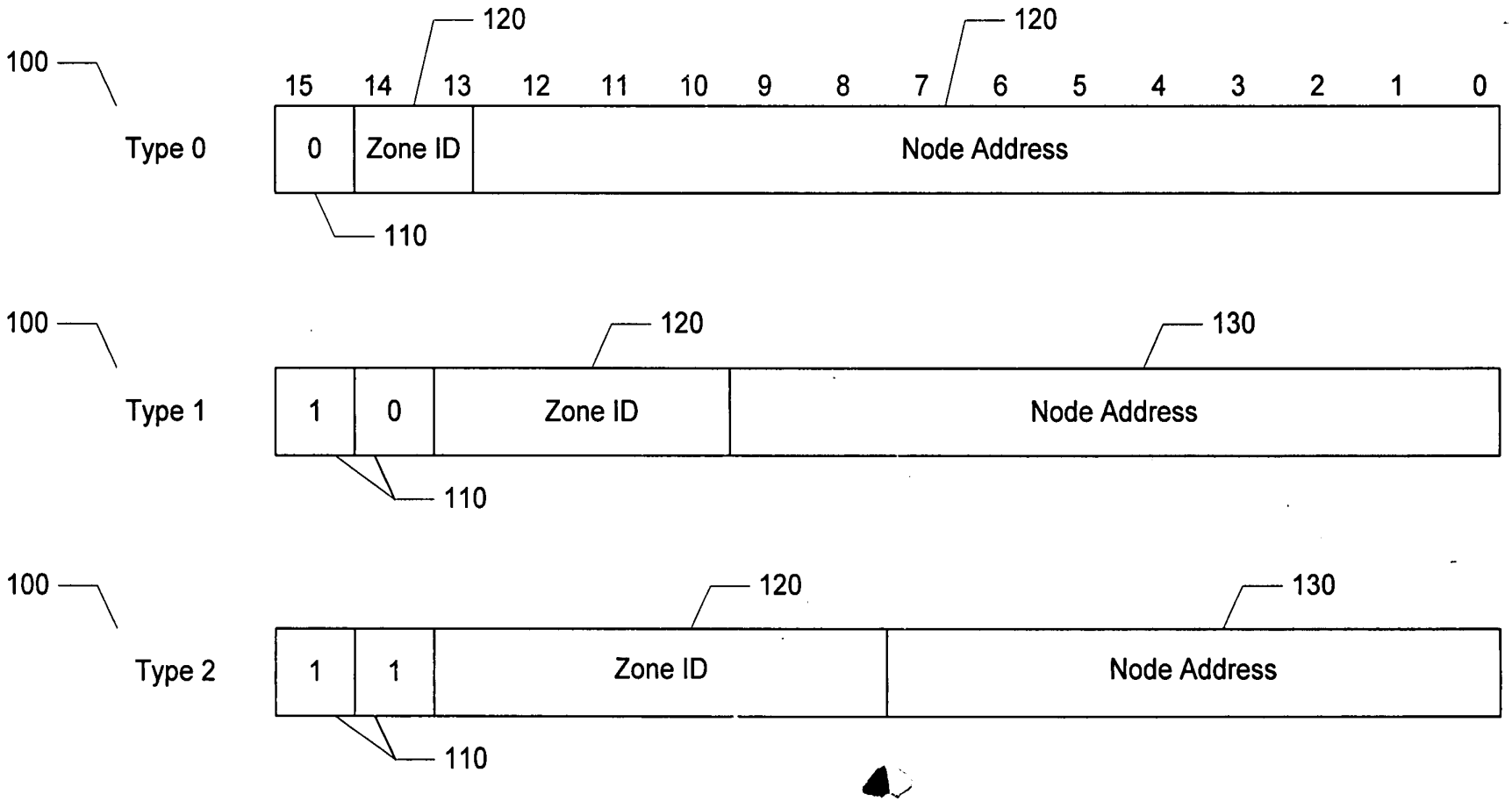

Fig. 1

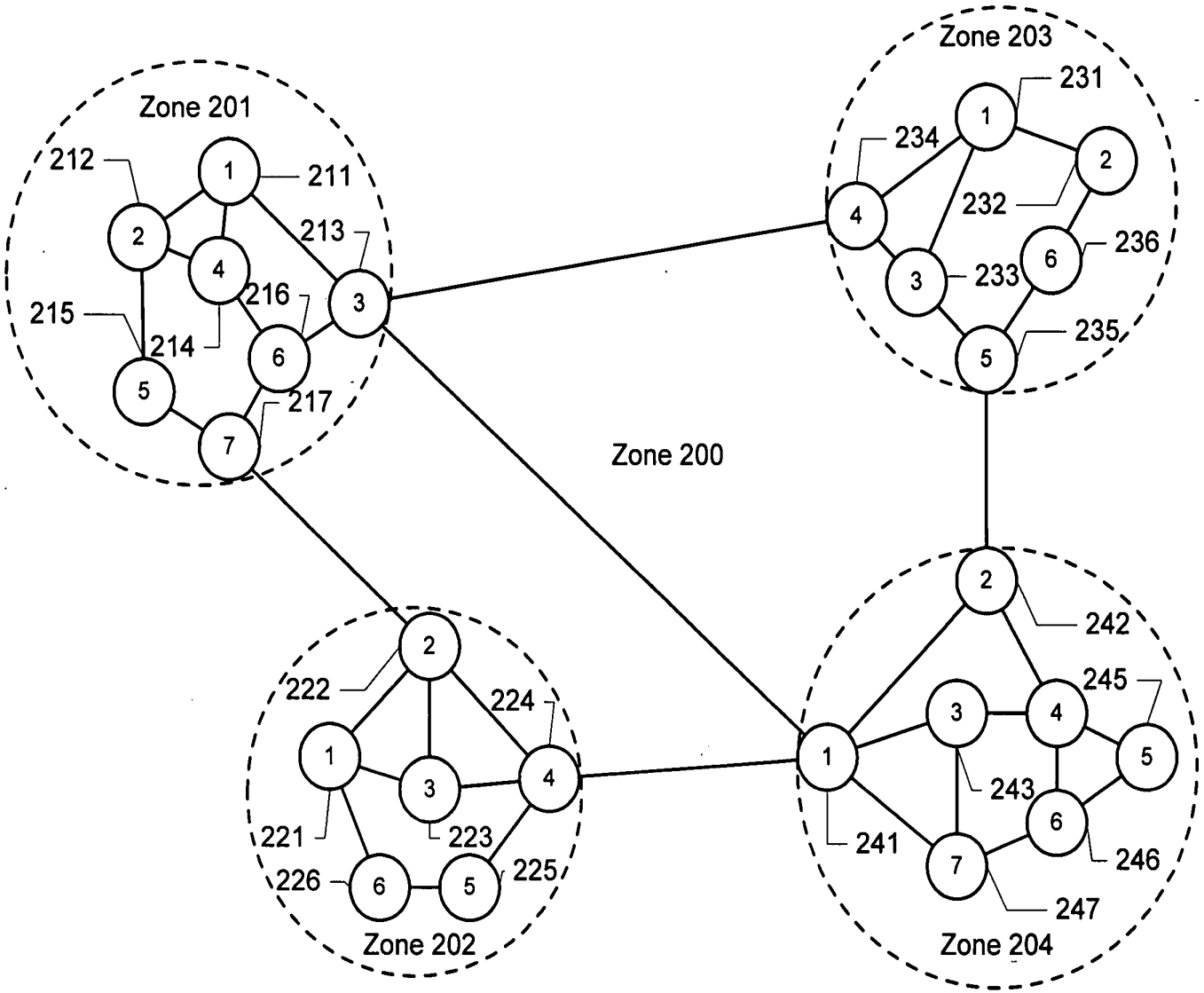

Fig. 2

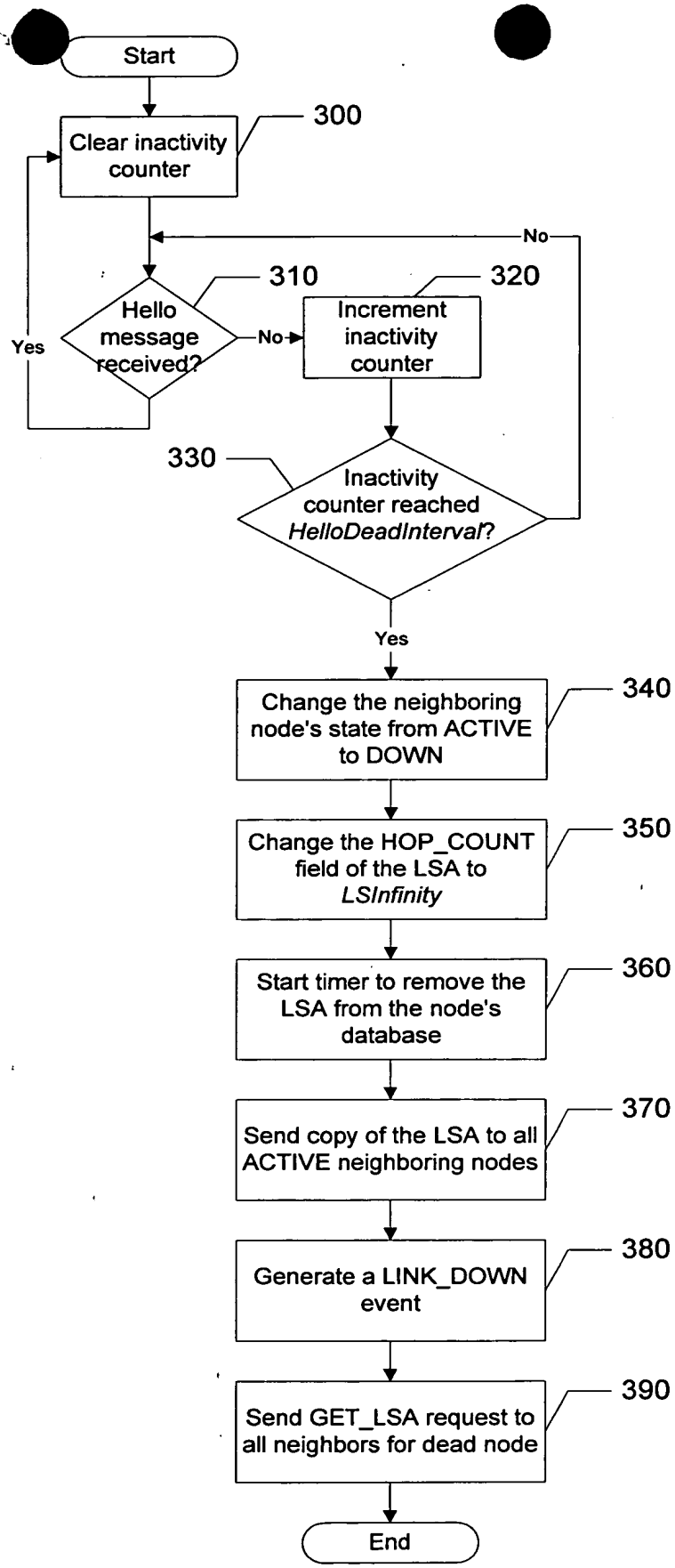
Start
Clear inactivity counter
300
No
310
320
Hello message received?
Yes
No
Increment inactivity counter
330
Inactivity counter reached HelloDeadInterval?
Yes
Change the neighboring node's state from ACTIVE to DOWN
340
Change the HOP_COUNT field of the LSA to LSInfinity
350
Start timer to remove the LSA from the node's database
360
Send copy of the LSA to all ACTIVE neighboring nodes
370
Generate a LINK_DOWN event
380
Send GET_LSA request to all neighbors for dead node
390
End

Fig. 3

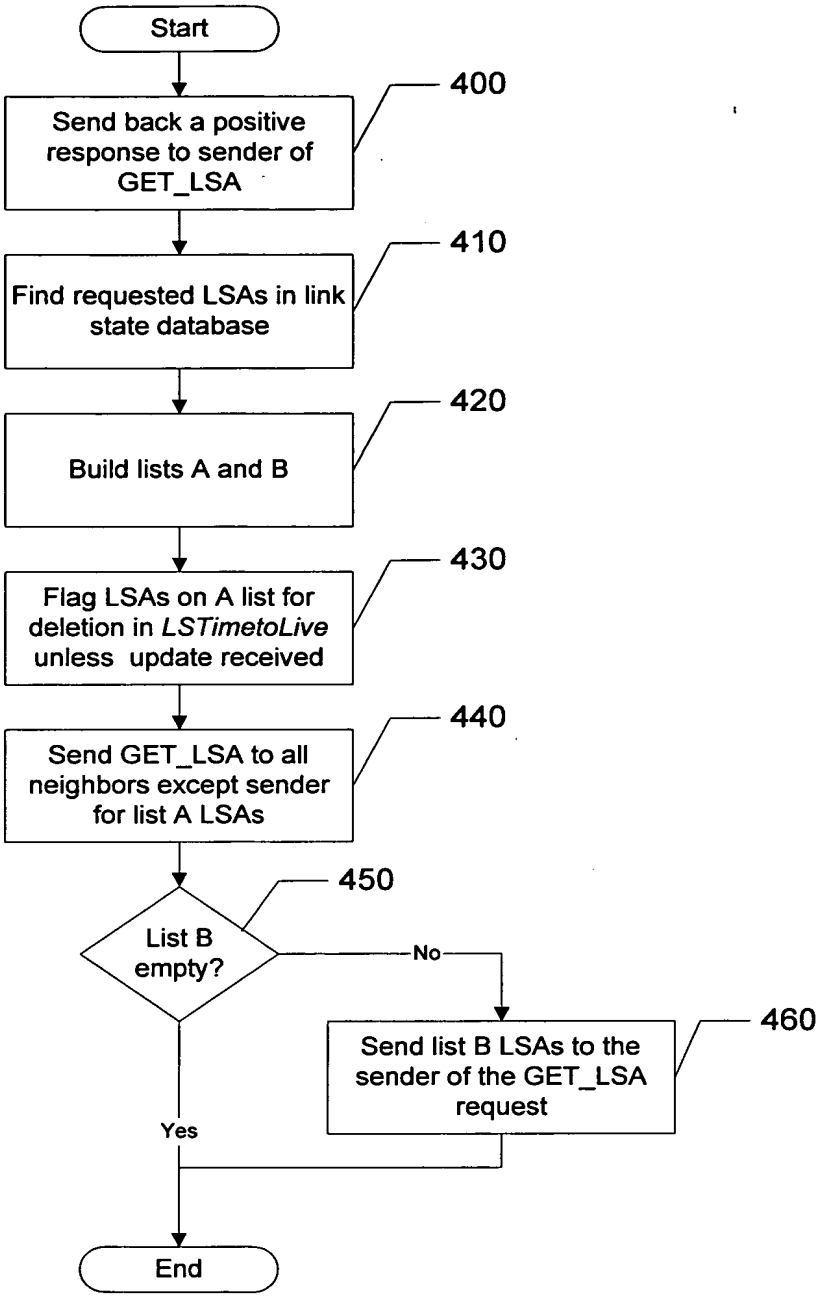
Start
Send back a positive response to sender of GET_LSA
400
Find requested LSAs in link state database
410
Build lists A and B
420
Flag LSAs on A list for deletion in *LSTimetoLive* unless update received
430
Send GET_LSA to all neighbors except sender for list A LSAs
440
List B empty?
450
No
Send list B LSAs to the sender of the GET_LSA request
460
Yes
End

Fig. 4

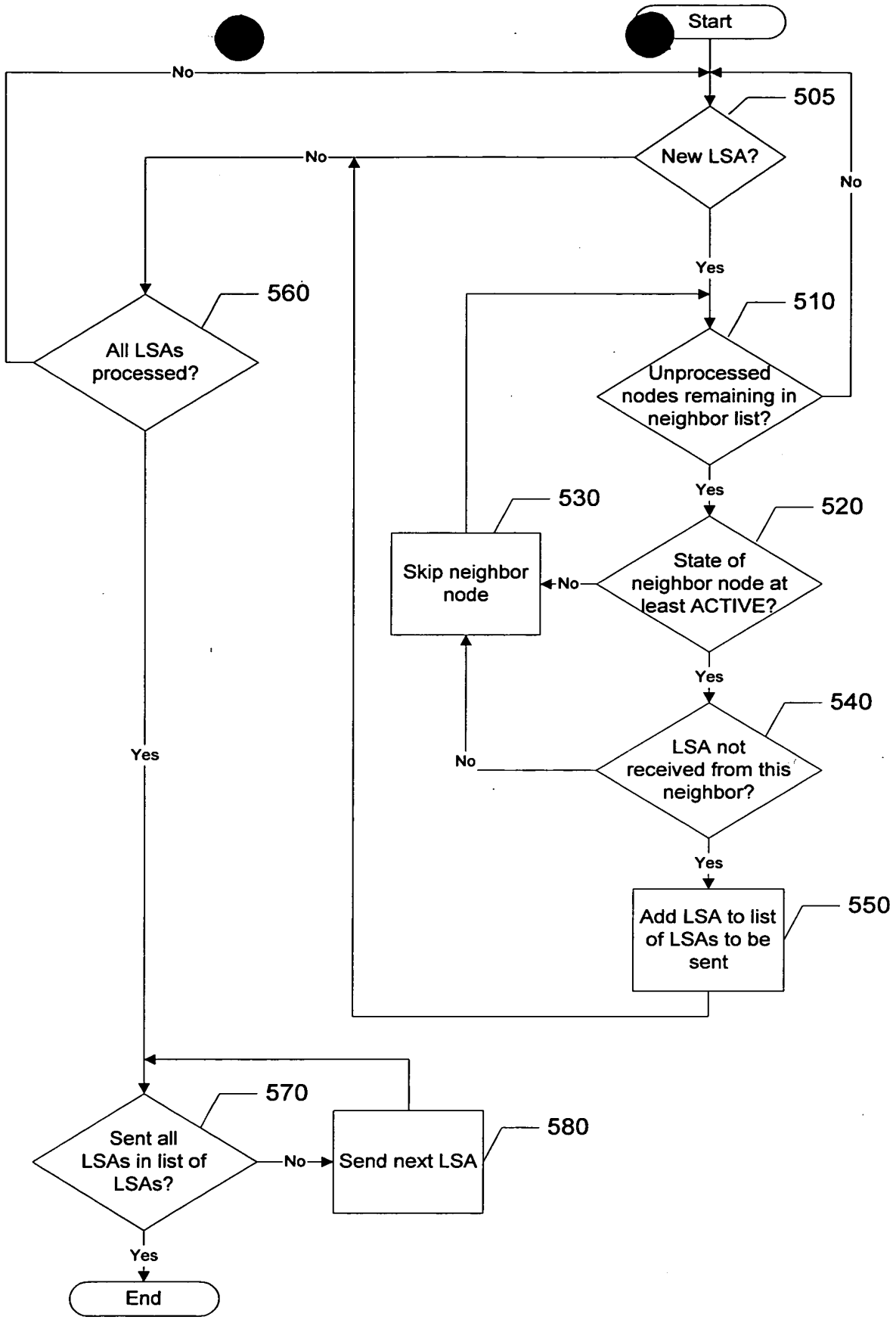
Start
505
New LSA?
No
Yes
No
510
Unprocessed nodes remaining in neighbor list?
Yes
520
State of neighbor node at least ACTIVE?
No
530
Skip neighbor node
Yes
540
LSA not received from this neighbor?
No
Yes
550
Add LSA to list of LSAs to be sent
560
All LSAs processed?
No
Yes
570
Sent all LSAs in list of LSAs?
No
580
Send next LSA
Yes
End

Fig. 5

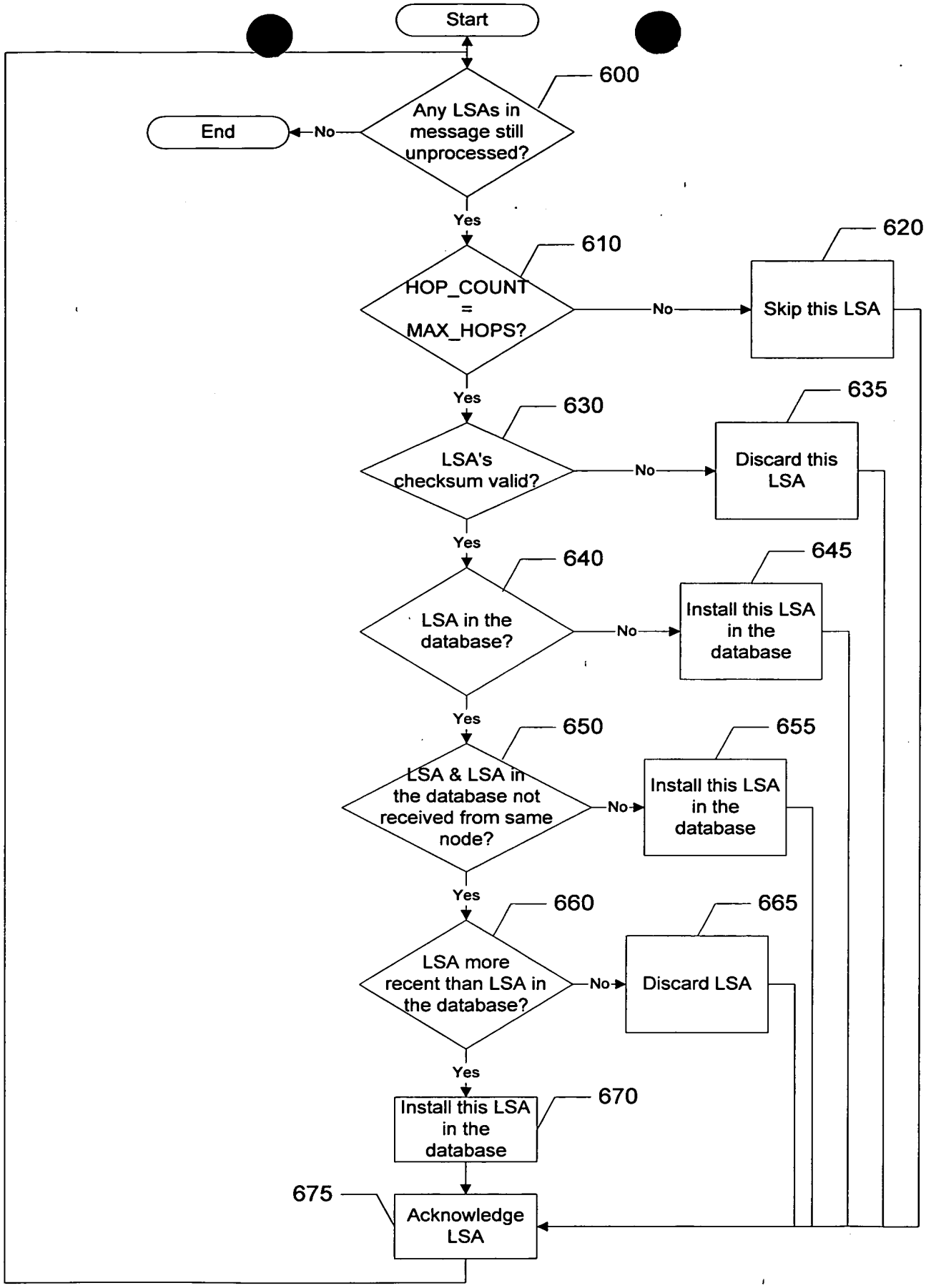
Start
600
Any LSAs in message still unprocessed?
No
End
Yes
610
HOP_COUNT = MAX_HOPS?
No
620
Skip this LSA
Yes
630
LSA's checksum valid?
No
635
Discard this LSA
Yes
640
LSA in the database?
No
645
Install this LSA in the database
Yes
650
LSA & LSA in the database not received from same node?
No
655
Install this LSA in the database
Yes
660
LSA more recent than LSA in the database?
No
665
Discard LSA
Yes
670
Install this LSA in the database
675
Acknowledge LSA

Fig. 6

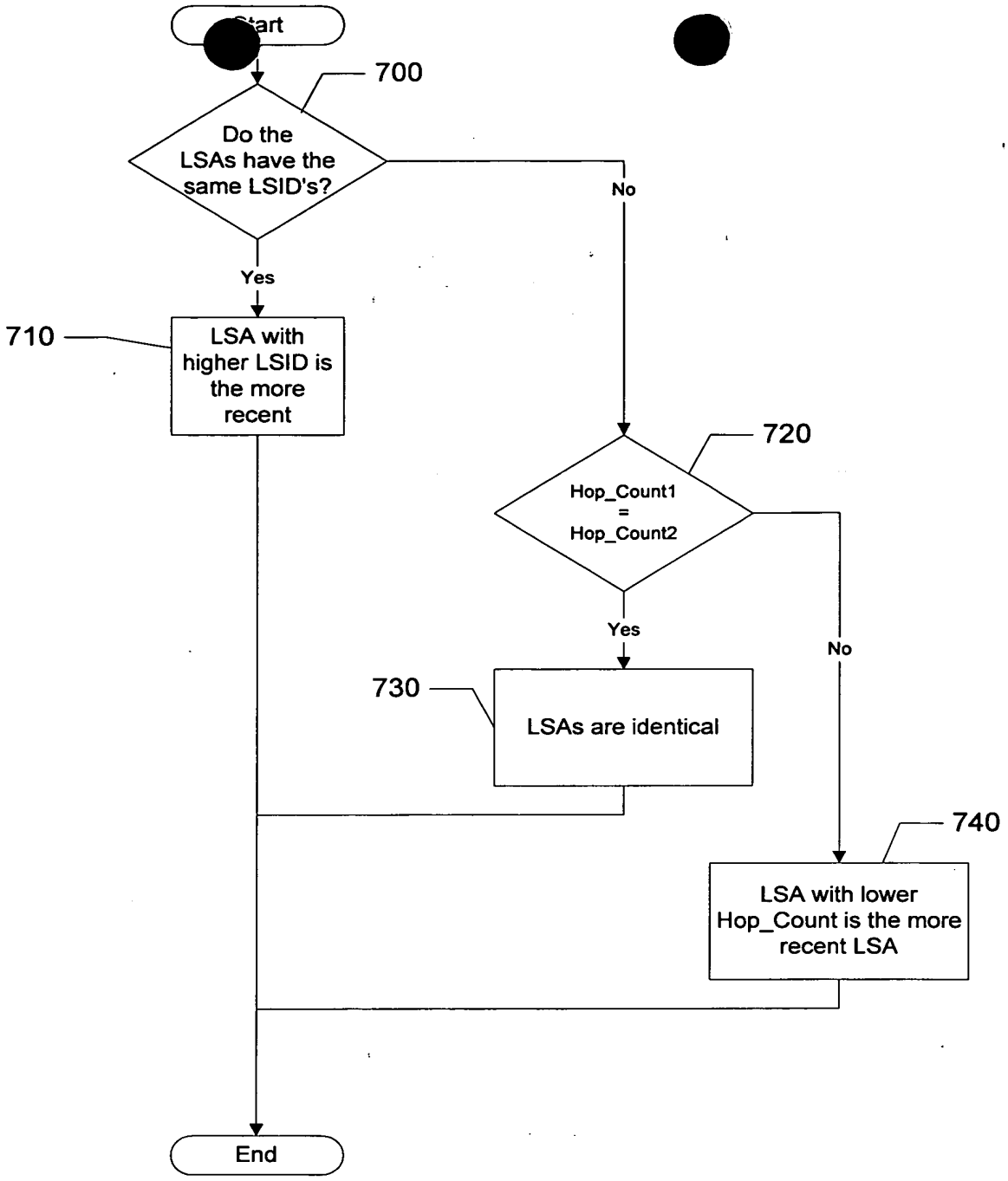
Start
700
Do the LSAs have the same LSID's?
No
Yes
710
LSA with higher LSID is the more recent
720
Hop_Count1 = Hop_Count2
Yes
No
730
LSAs are identical
740
LSA with lower Hop_Count is the more recent LSA
End

Fig. 7

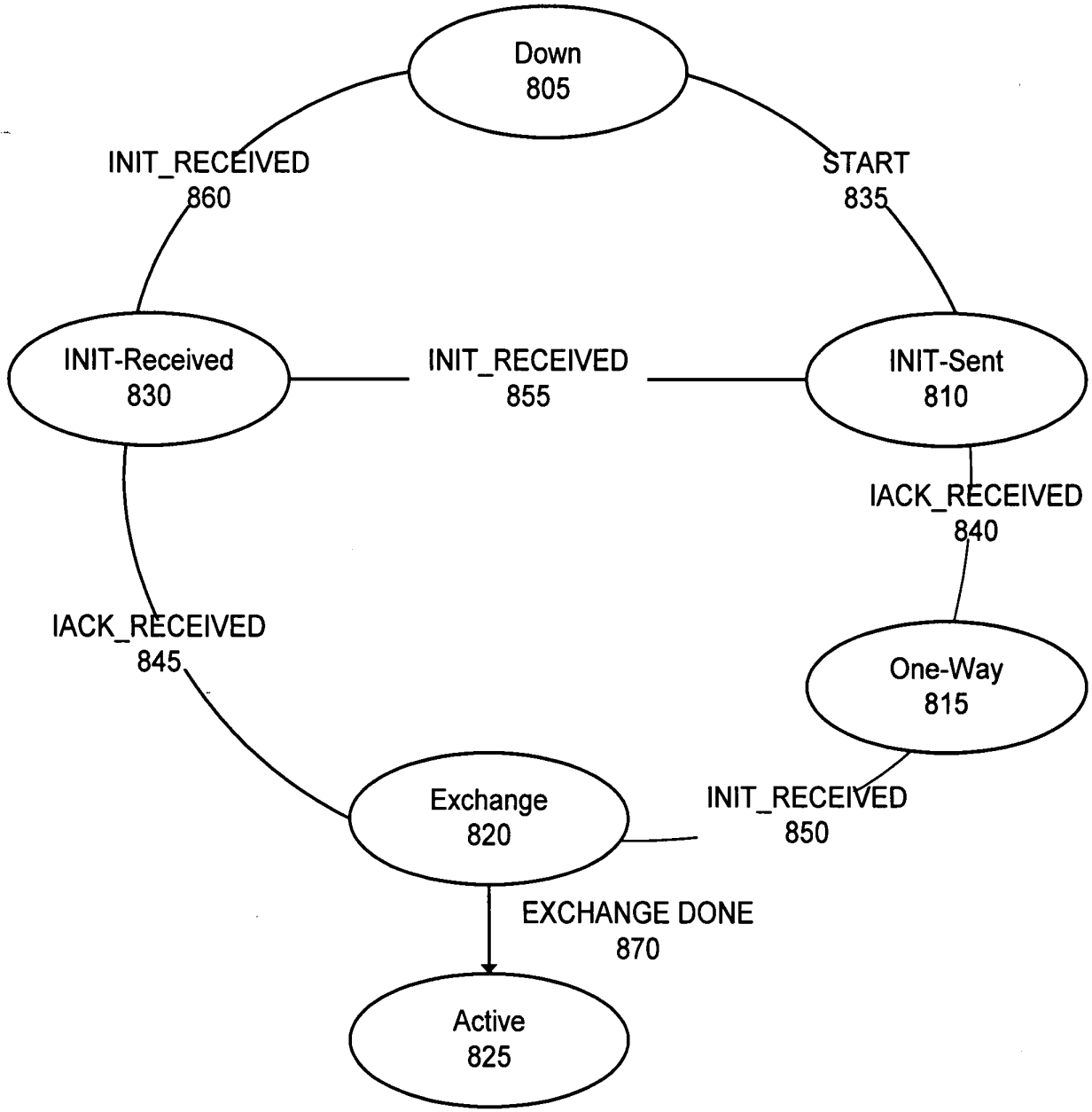

Fig. 8

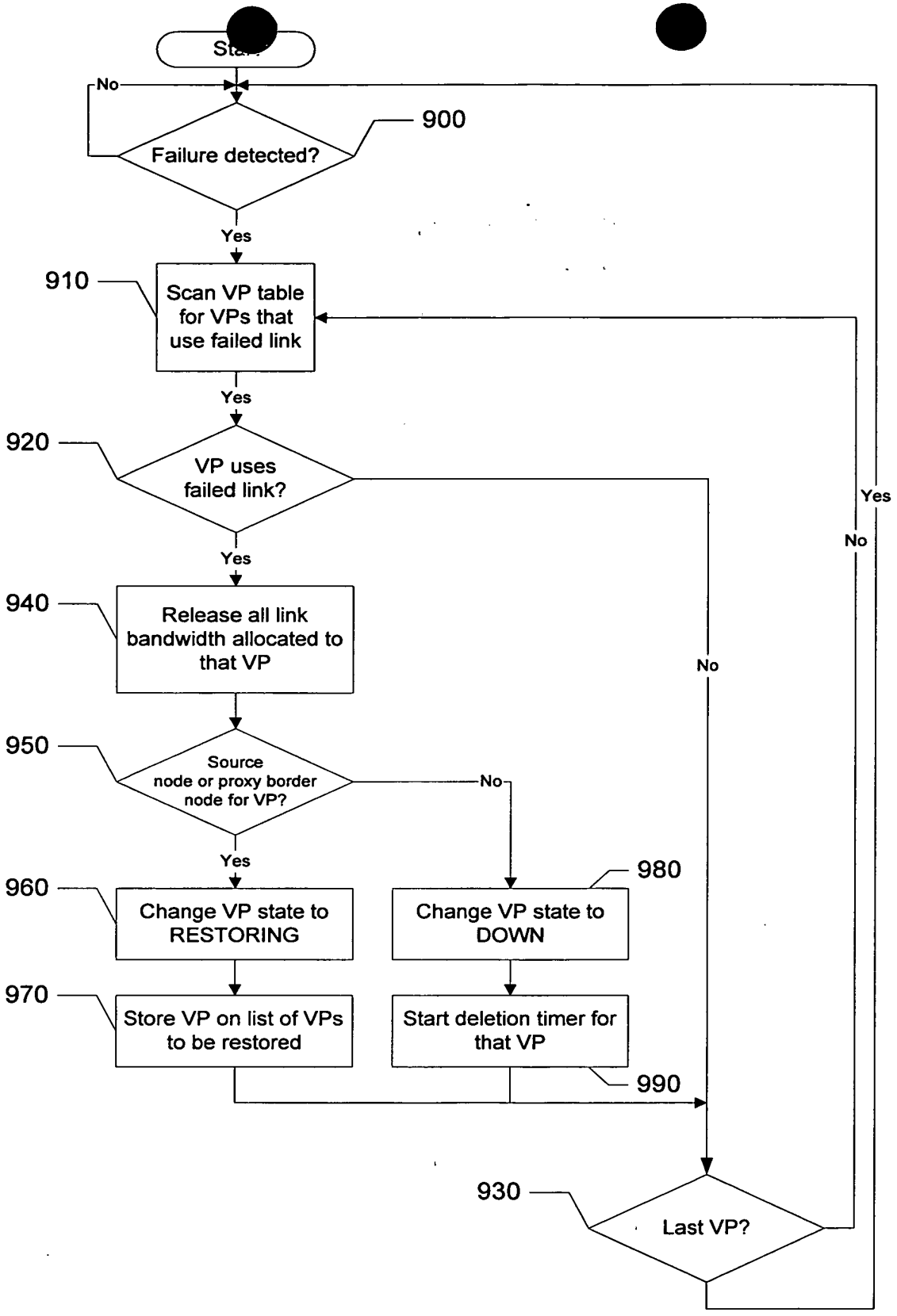
No
900
Failure detected?
Yes
910
Scan VP table for VPs that use failed link
Yes
920
VP uses failed link?
Yes
940
Release all link bandwidth allocated to that VP
950
Source node or proxy border node for VP?
No
Yes
960
Change VP state to RESTORING
980
Change VP state to DOWN
970
Store VP on list of VPs to be restored
Start deletion timer for that VP
990
No
Yes
930
Last VP?
No
Yes

Fig. 9

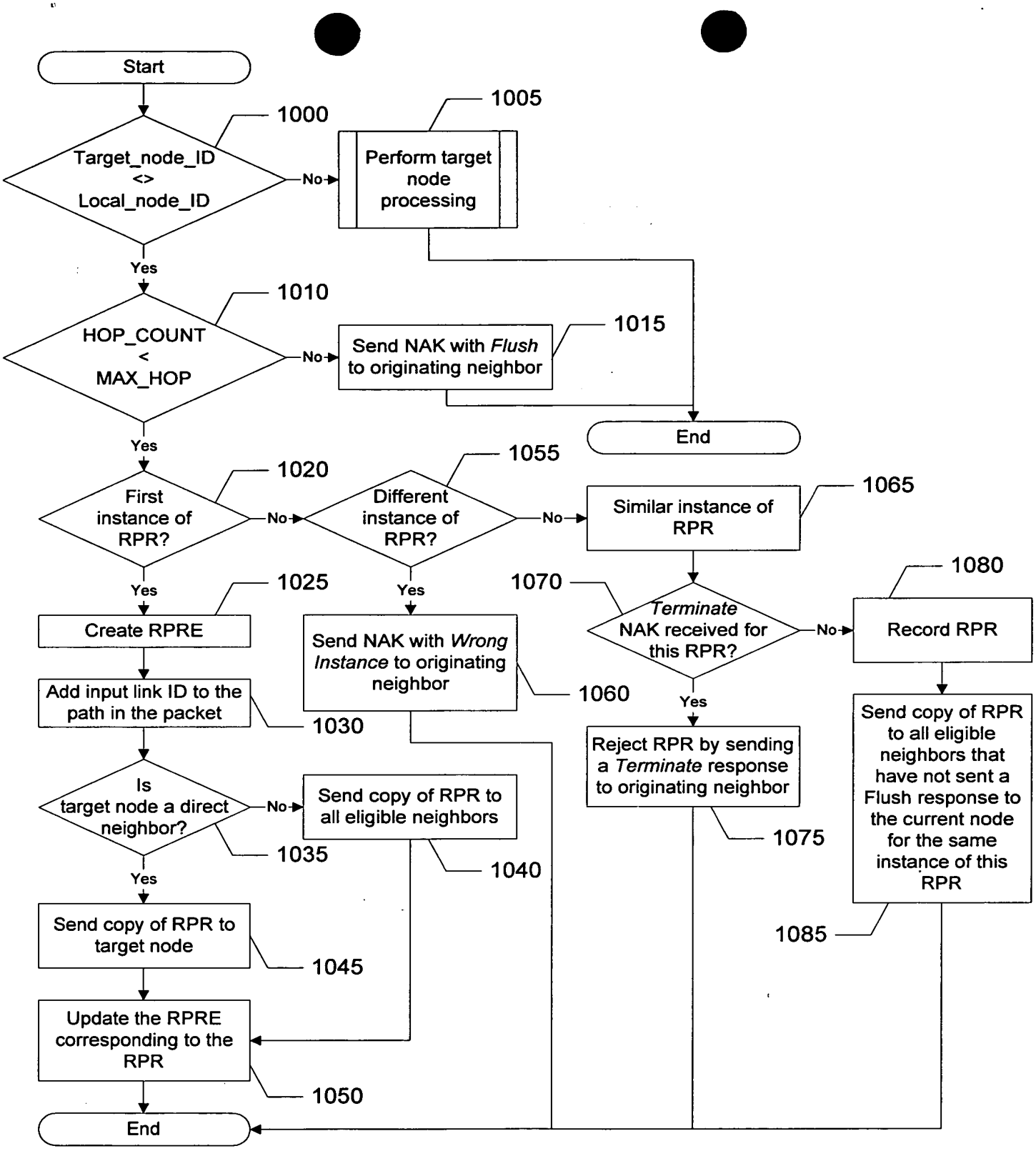
Start
1000
Target_node_ID <> Local_node_ID
No
1005
Perform target node processing
Yes
1010
HOP_COUNT < MAX_HOP
No
1015
Send NAK with *Flush* to originating neighbor
End
Yes
1020
First instance of RPR?
No
1055
Different instance of RPR?
No
1065
Similar instance of RPR
Yes
1025
Create RPRE
Yes
Send NAK with *Wrong Instance* to originating neighbor
1060
1070
Terminate NAK received for this RPR?
No
1080
Record RPR
Yes
Reject RPR by sending a *Terminate* response to originating neighbor
1075
Send copy of RPR to all eligible neighbors that have not sent a Flush response to the current node for the same instance of this RPR
1085
Add input link ID to the path in the packet
1030
Is target node a direct neighbor?
No
Send copy of RPR to all eligible neighbors
1035
1040
Yes
Send copy of RPR to target node
1045
Update the RPRE corresponding to the RPR
1050
End

Fig. 10

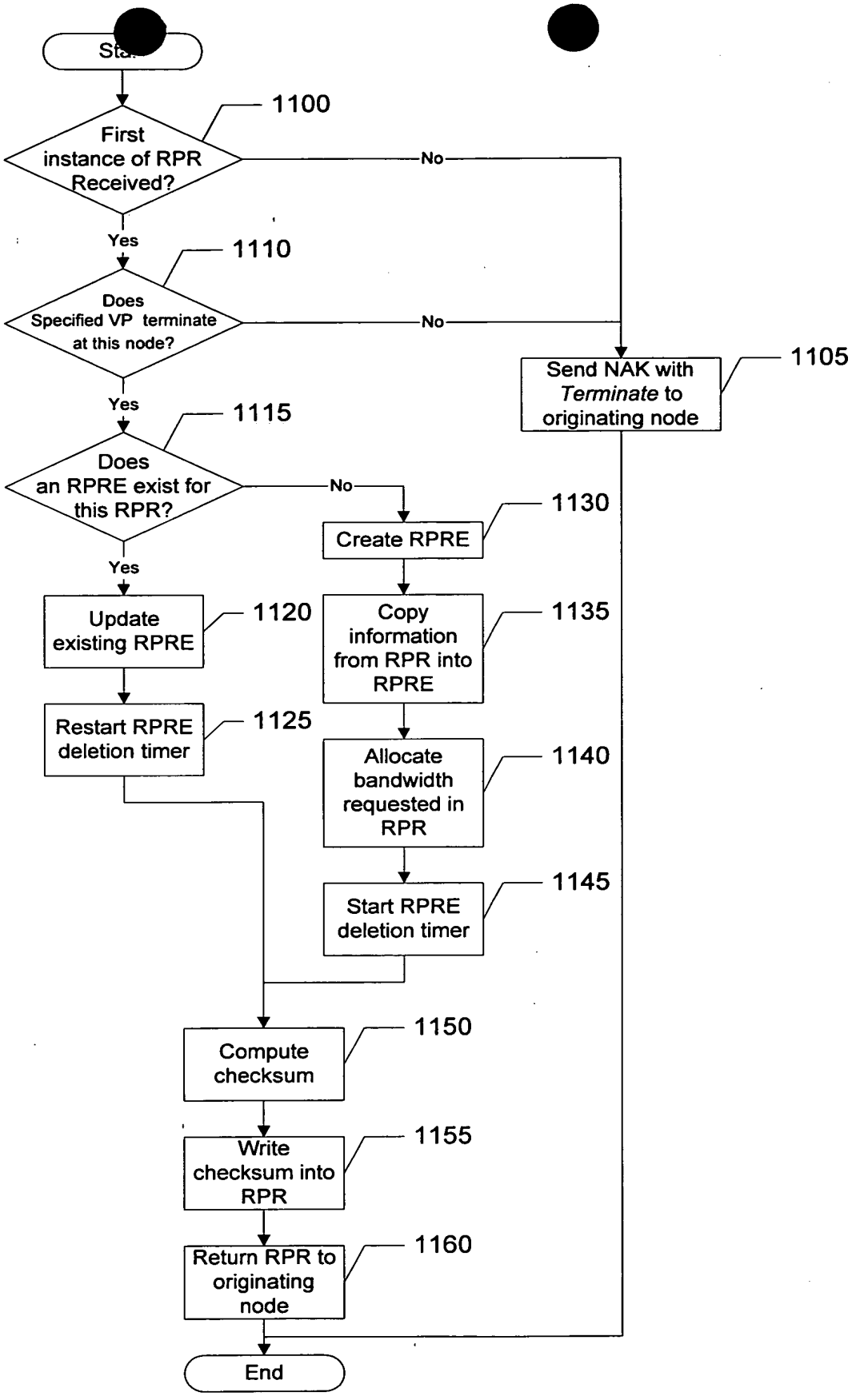
Start
1100
First instance of RPR Received?
No
Yes
1110
Does Specified VP terminate at this node?
No
Yes
1105
Send NAK with *Terminate* to originating node
1115
Does an RPRE exist for this RPR?
No
Yes
1130
Create RPRE
1120
Update existing RPRE
1135
Copy information from RPR into RPRE
1125
Restart RPRE deletion timer
1140
Allocate bandwidth requested in RPR
1145
Start RPRE deletion timer
1150
Compute checksum
1155
Write checksum into RPR
1160
Return RPR to originating node
End

Fig. 11

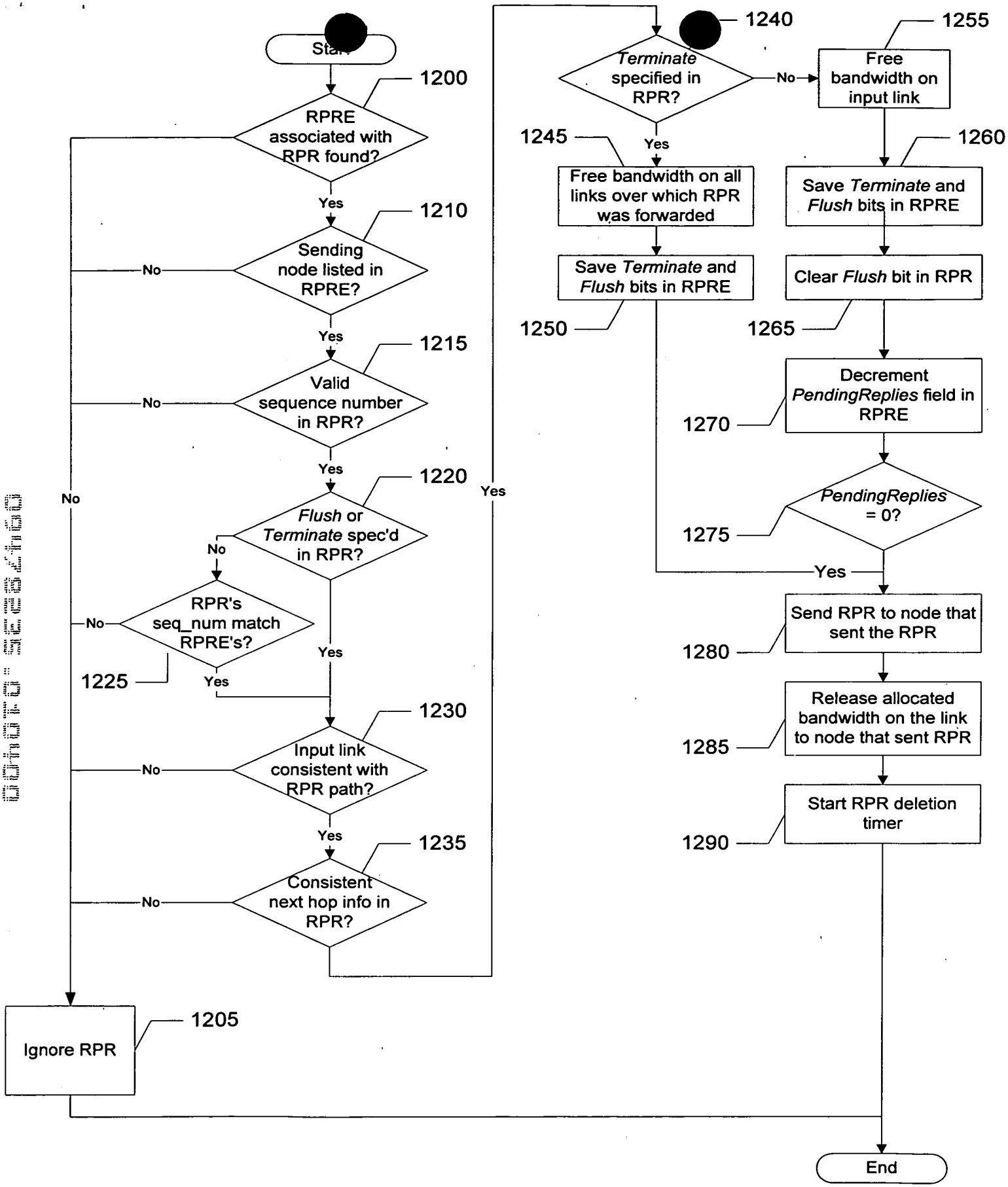
Start
1200
RPRE associated with RPR found?
Yes
No
1210
Sending node listed in RPRE?
Yes
No
1215
Valid sequence number in RPR?
Yes
No
1220
Flush or Terminate spec'd in RPR?
No
Yes
RPR's seq_num match RPRE's?
No
Yes
1225
1230
Input link consistent with RPR path?
No
Yes
1235
Consistent next hop info in RPR?
No
Yes
1205
Ignore RPR
1240
Terminate specified in RPR?
No
Yes
1245
Free bandwidth on all links over which RPR was forwarded
Save Terminate and Flush bits in RPRE
1250
1255
Free bandwidth on input link
1260
Save Terminate and Flush bits in RPRE
Clear Flush bit in RPR
1265
Decrement PendingReplies field in RPRE
1270
PendingReplies = 0?
1275
Yes
Send RPR to node that sent the RPR
1280
Release allocated bandwidth on the link to node that sent RPR
1285
Start RPR deletion timer
1290
End

Fig. 12

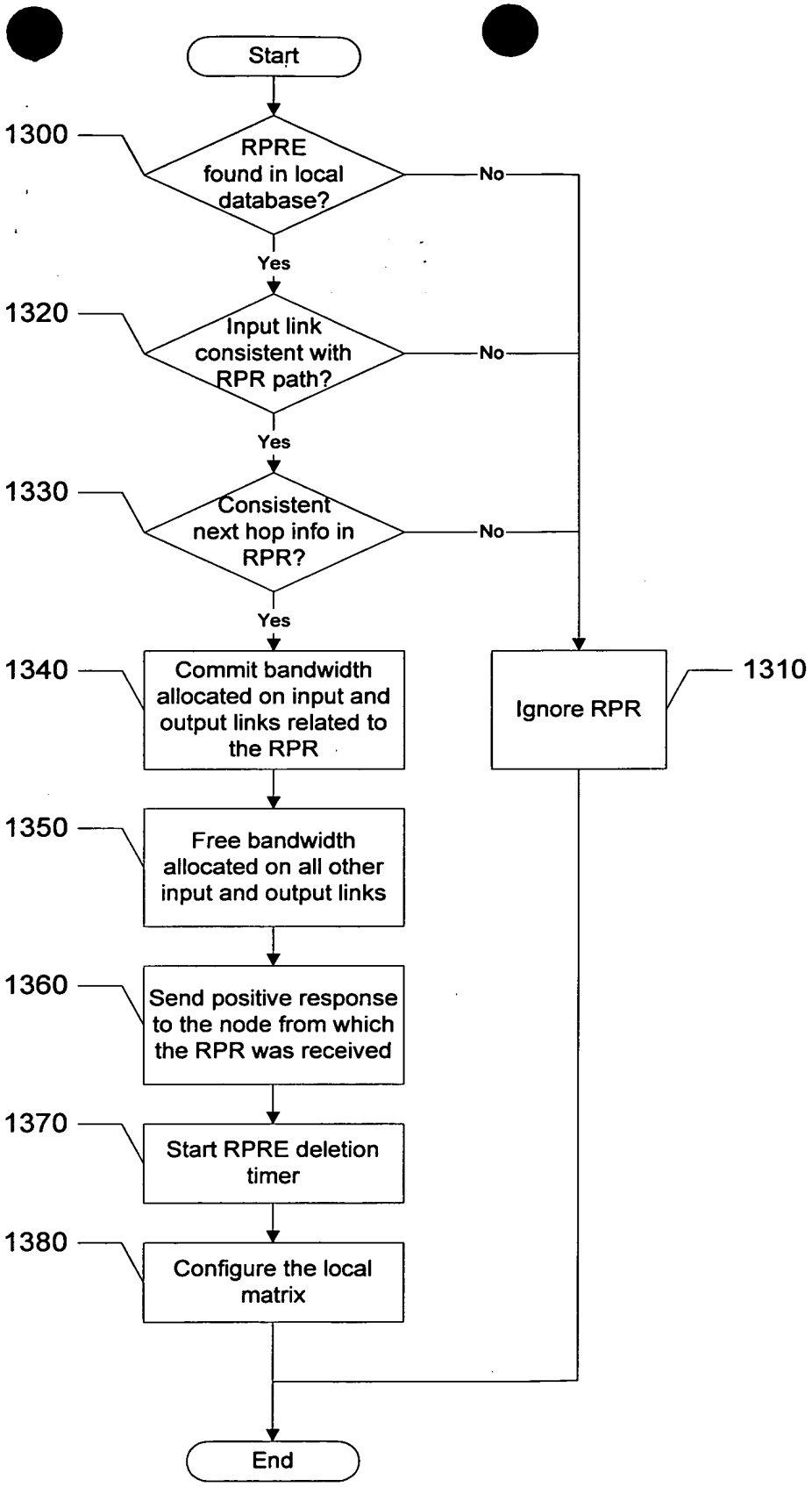
Start
1300
RPRE found in local database?
No
Yes
1320
Input link consistent with RPR path?
No
Yes
1330
Consistent next hop info in RPR?
No
Yes
1340
Commit bandwidth allocated on input and output links related to the RPR
1350
Free bandwidth allocated on all other input and output links
1360
Send positive response to the node from which the RPR was received
1370
Start RPRE deletion timer
1380
Configure the local matrix
Ignore RPR
1310
End

Fig. 13

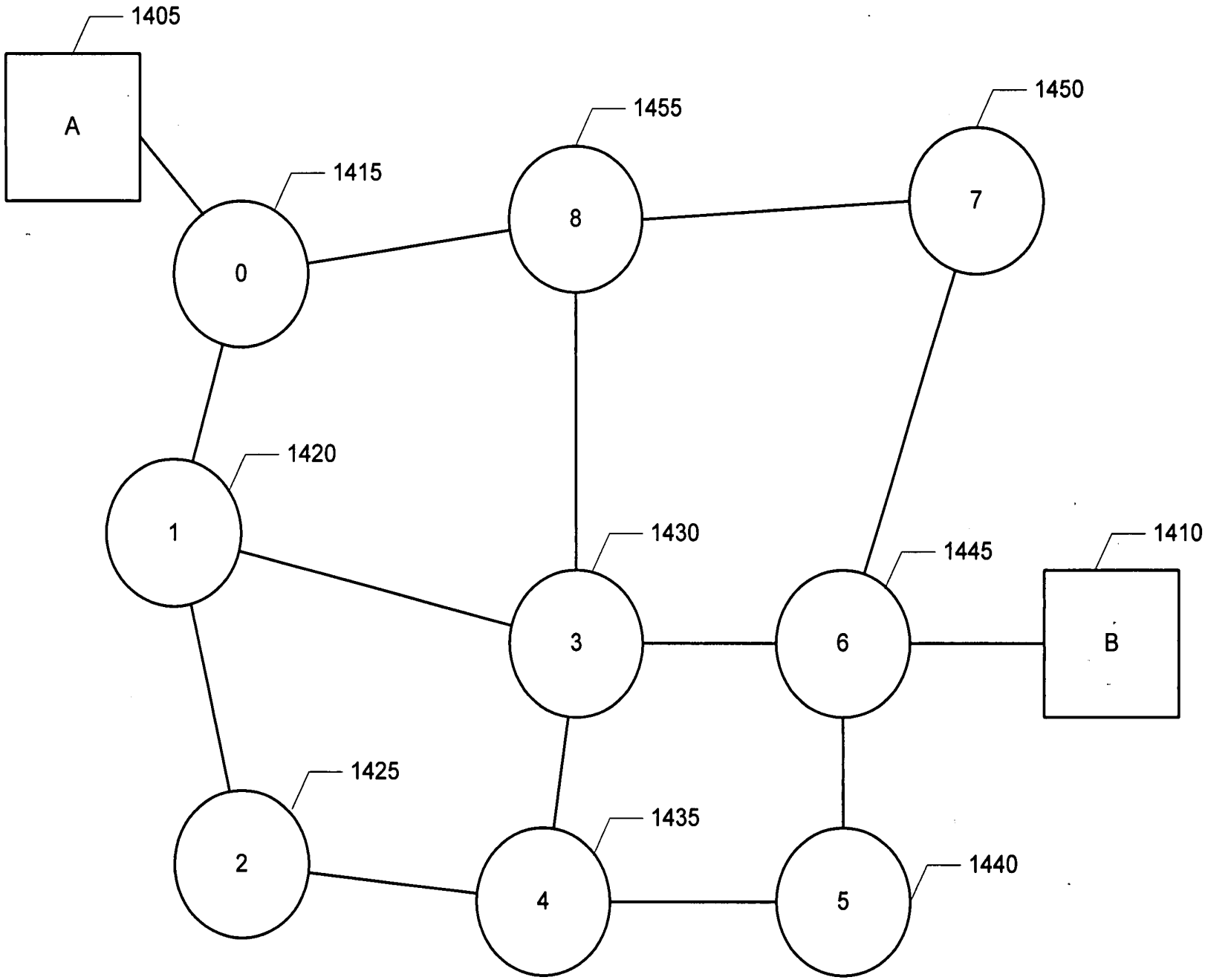

Fig. 14

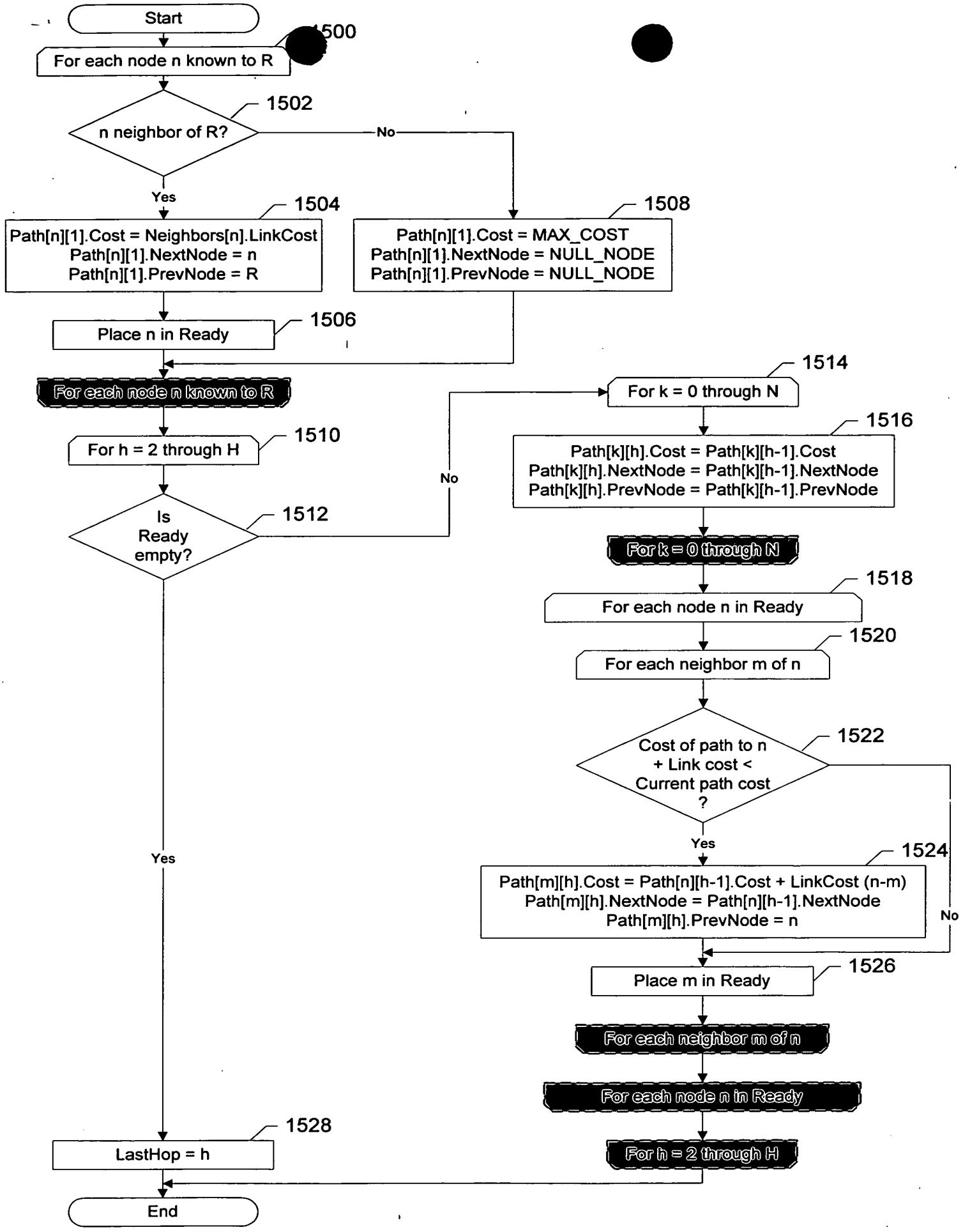
Start
For each node n known to R
1500
n neighbor of R?
1502
No
Yes
Path[n][1].Cost = Neighbors[n].LinkCost
Path[n][1].NextNode = n
Path[n][1].PrevNode = R
1504
Path[n][1].Cost = MAX_COST
Path[n][1].NextNode = NULL_NODE
Path[n][1].PrevNode = NULL_NODE
1508
Place n in Ready
1506
For each node n known to R
For h = 2 through H
1510
Is Ready empty?
1512
No
Yes
For k = 0 through N
1514
Path[k][h].Cost = Path[k][h-1].Cost
Path[k][h].NextNode = Path[k][h-1].NextNode
Path[k][h].PrevNode = Path[k][h-1].PrevNode
1516
For k = 0 through N
For each node n in Ready
1518
For each neighbor m of n
1520
Cost of path to n + Link cost < Current path cost ?
1522
Yes
No
Path[m][h].Cost = Path[n][h-1].Cost + LinkCost (n-m)
Path[m][h].NextNode = Path[n][h-1].NextNode
Path[m][h].PrevNode = n
1524
Place m in Ready
1526
For each neighbor m of n
For each node n in Ready
For h = 2 through H
LastHop = h
1528
End

Fig. 15A

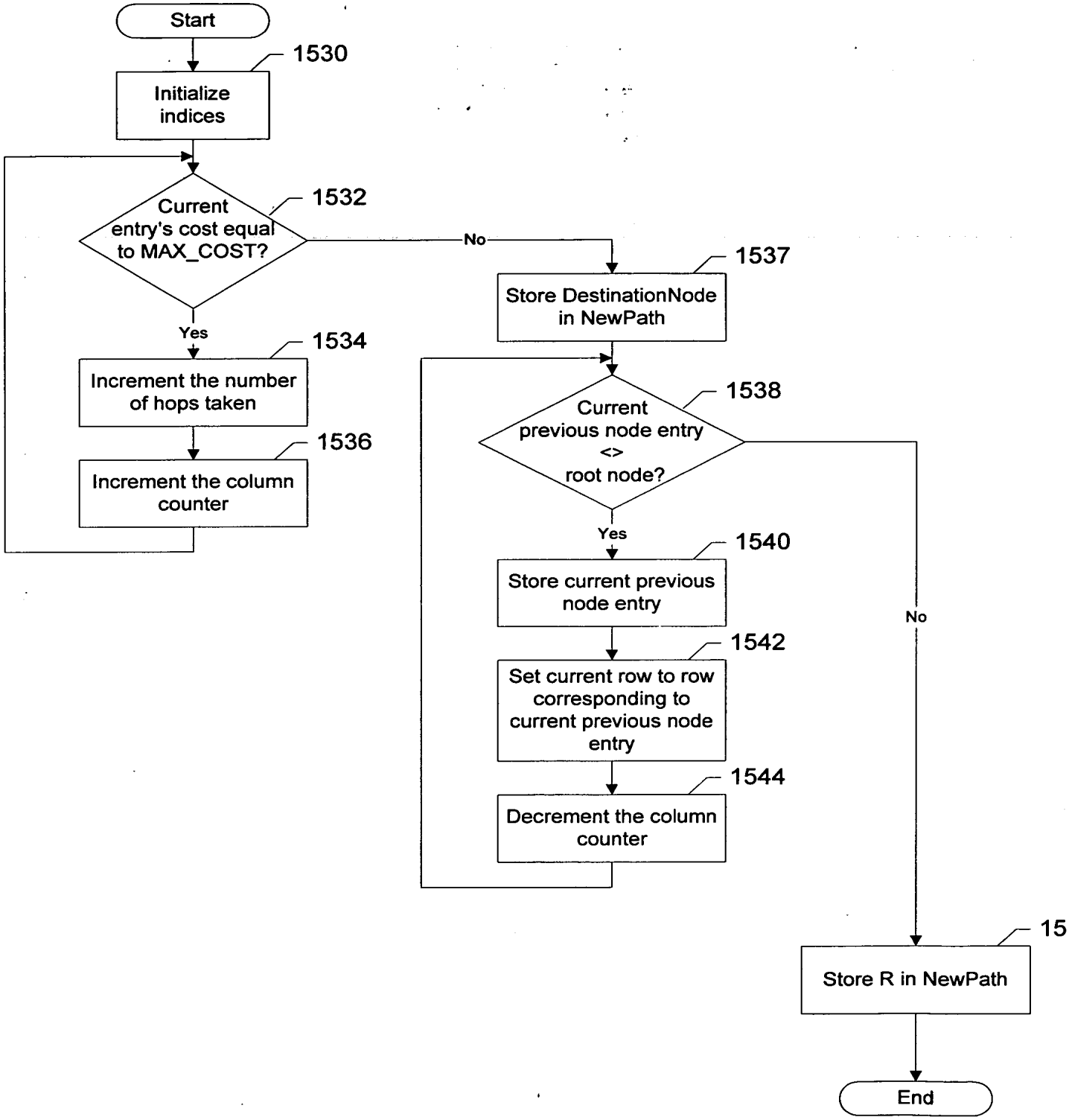
Start
1530
Initialize indices
1532
Current entry's cost equal to MAX_COST?
No
Yes
1534
Increment the number of hops taken
1536
Increment the column counter
1537
Store DestinationNode in NewPath
1538
Current previous node entry <> root node?
Yes
1540
Store current previous node entry
1542
Set current row to row corresponding to current previous node entry
1544
Decrement the column counter
No
15
Store R in NewPath
End

Fig. 15B

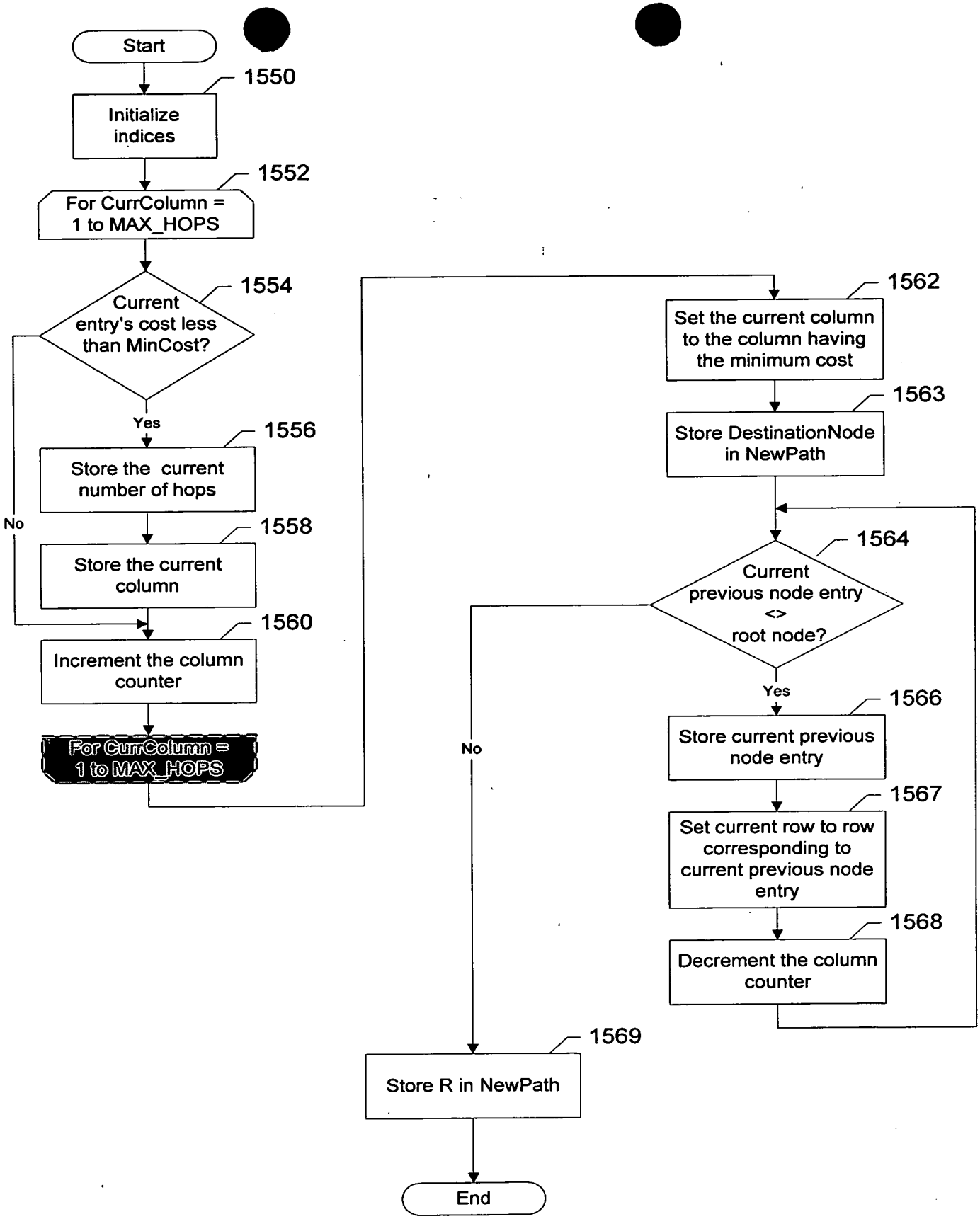
Start
1550
Initialize indices
1552
For CurrColumn = 1 to MAX_HOPS
1554
Current entry's cost less than MinCost?
Yes
No
1556
Store the current number of hops
1558
Store the current column
1560
Increment the column counter
For CurrColumn = 1 to MAX_HOPS
1562
Set the current column to the column having the minimum cost
1563
Store DestinationNode in NewPath
1564
Current previous node entry <> root node?
Yes
No
1566
Store current previous node entry
1567
Set current row to row corresponding to current previous node entry
1568
Decrement the column counter
1569
Store R in NewPath
End

Fig. 15C

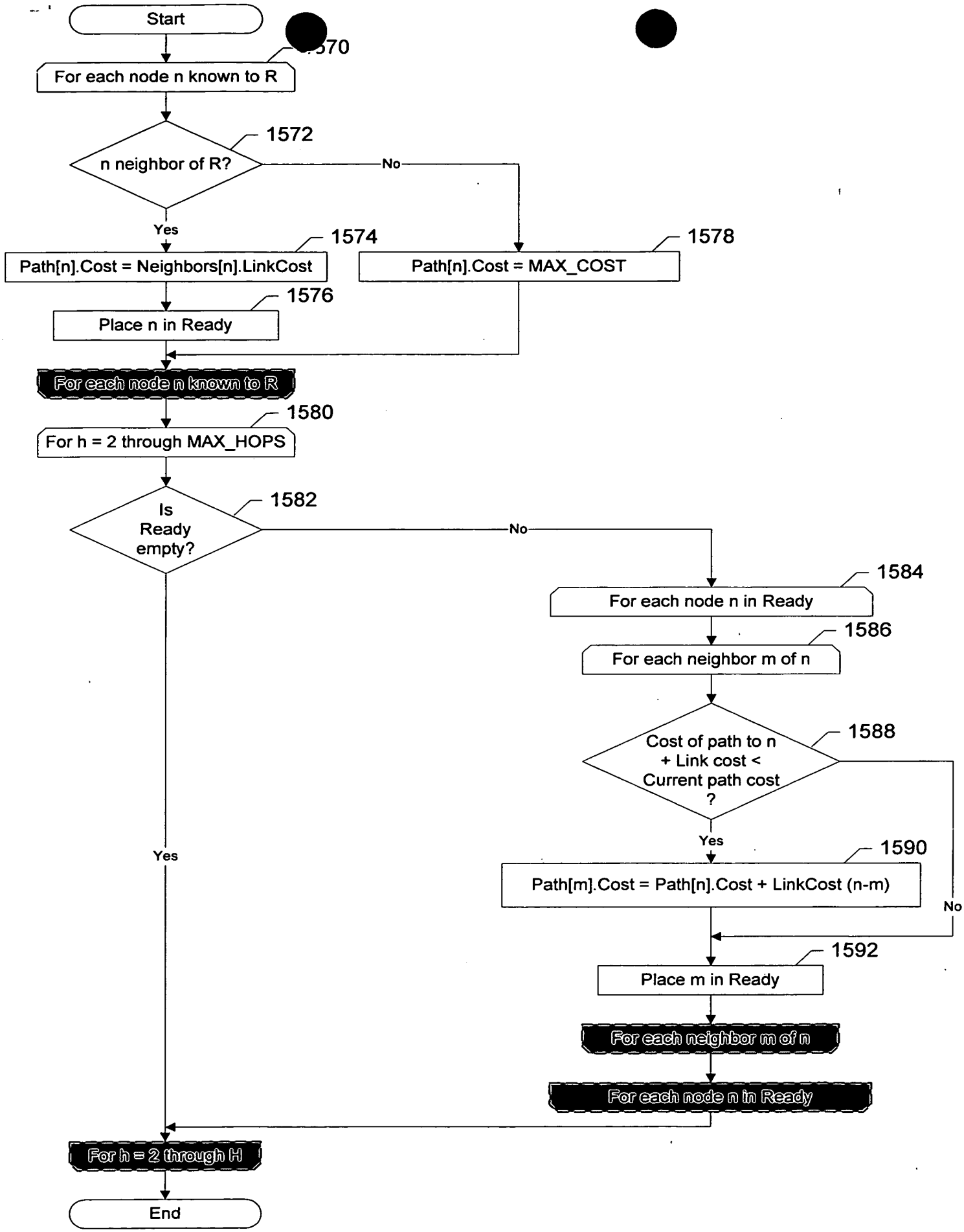
Start
For each node n known to R
1572
n neighbor of R?
No
Yes
1574
Path[n].Cost = Neighbors[n].LinkCost
1578
Path[n].Cost = MAX_COST
1576
Place n in Ready
For each node n known to R
1580
For h = 2 through MAX_HOPS
1582
Is Ready empty?
No
1584
For each node n in Ready
1586
For each neighbor m of n
1588
Cost of path to n + Link cost < Current path cost ?
Yes
1590
Path[m].Cost = Path[n].Cost + LinkCost (n-m)
No
1592
Place m in Ready
For each neighbor m of n
For each node n in Ready
Yes
For h = 2 through H
End

Fig. 15D